Lab 11: Random Number Generation

Taaruni Ananya | Session 008 | CRN: 15623

04/18/2025

Dr. Lan Gao

Introduction

Lab 12 is centered around programming with RNG (random number generators), which involves practicing writing code with RISC-V, VSCode, and Terminal as well as practicing using FileZilla and RARS .jar. Lab 12 looks into exploring how to use loops to write a program that can take in a given value, loop through, and produce the needed amount of values; as the parts progress, one of the parts even delves into cutting off numbers and having a range for outputted values. The lab explains how to use Terminal, VSCode, and RISC-V. The lab also gives hands-on experience with writing a program to practice looping as well as generating random numbers and patterns..

Purpose

The purpose of this lab is to learn how to program efficiently as well as the use of loops for printing randomly generated numbers under certain circumstances given by the programmer, along with the commands and practice writing RISC-V utilizing VSCode; this lab also works to use PRNGs and loops to to cut off a certain length for each result as well as a range for the returned values. This lab is meant to experiment with using RISC-V to take in certain parameters and return output with programs and explore different ways to use these commands.

Main Objectives

- Learn how to seed an RNG
- Learn how to get a random value
- Learn how to seed the RNG to give an unpredictable sequence of values
- Learn how to make a random function
- Learn how to make a seed function
- Learn how to make the values smaller with a range
- Learn about random-looking seeds
- Reinforce learning of loops in Assembly

Brief Explanation

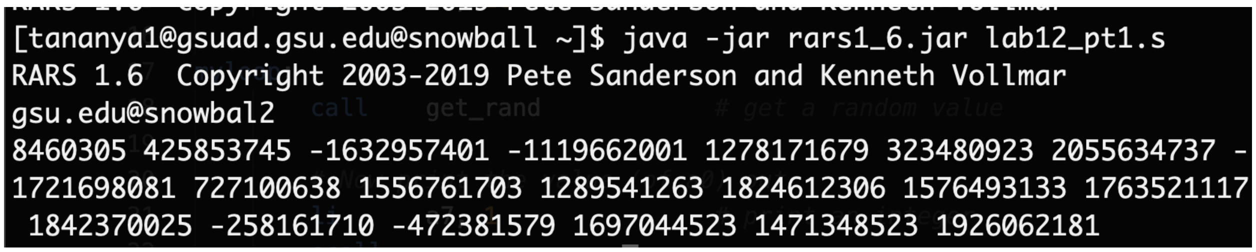
VS Code is an integrated development environment (IDE) where Venus and RISC-V can be accessed and programmed. "RISC-V is an open-source instruction set architecture used to develop custom processors for a variety of applications, from embedded designs to supercomputers." according to Synopsys.. "Terminal is an app for advanced users and developers that lets you communicate with the Mac operating system using a command line interface (CLI)." (Apple). Using VSCode, a programmer can use RISC-V to take in inputs, perform operations, and return outputs. Terminal provides a place to run programs that can output errors, what's causing them, what line, and which column to better assist programmers.

Methodology/Procedure

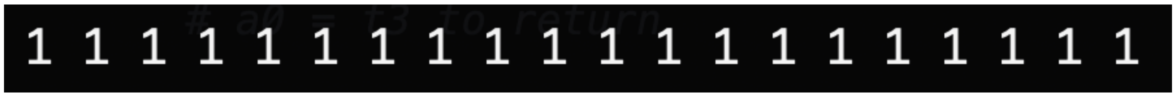
The procedure starts with adapting a program to test and learn more about how the program loops to return certain randomly generated numbers. Next, the code is completed to make a random function then tested–which resulted in randomly generated values. Then, the code is reused again to make a seed function. This program then looks deeper into using commands like 0x1f and 0x03 to make the values smaller. The program is also modified to have a smaller range as well as length of characters returned. Finally, the last part explores a random-looking seed with an unpredictable value and observed.

.

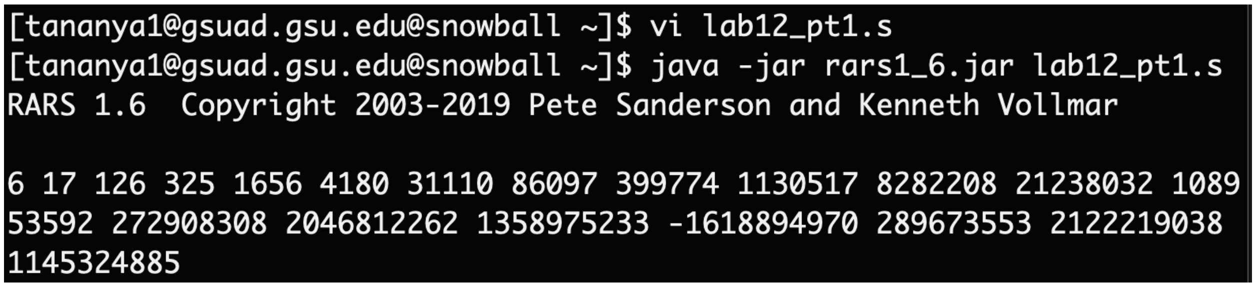
Documentation



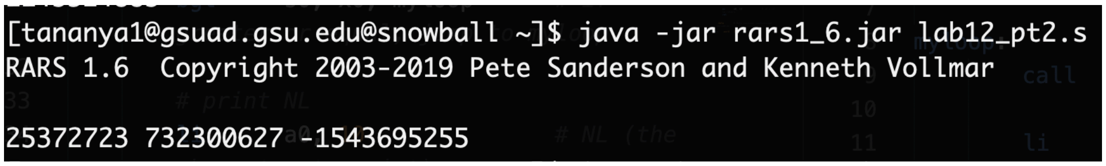
```
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt1.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar
gsu.edu@snowbal2
8460305 425853745 -1632957401 -1119662001 1278171679 323480923 2055634737 -
1721698081 727100638 1556761703 1289541263 1824612306 1576493133 1763521117
 1842370025 -258161710 -472381579 1697044523 1471348523 1926062181
```



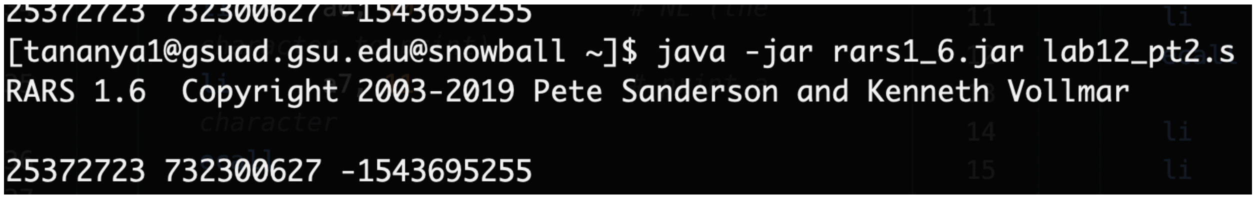
```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```



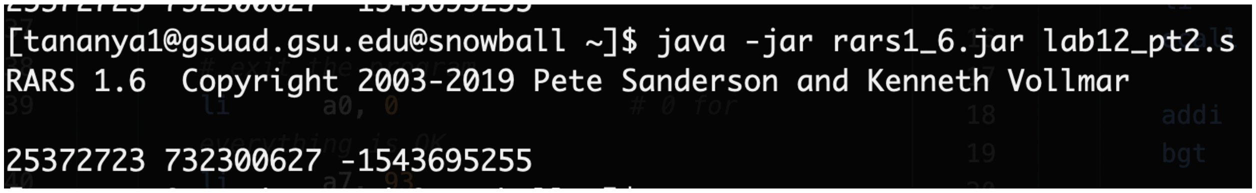
```
[tananya1@gsuad.gsu.edu@snowball ~]$ vi lab12_pt1.s
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt1.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

6 17 126 325 1656 4180 31110 86097 399774 1130517 8282208 21238032 1089
53592 272908308 2046812262 1358975233 -1618894970 289673553 2122219038
1145324885
```



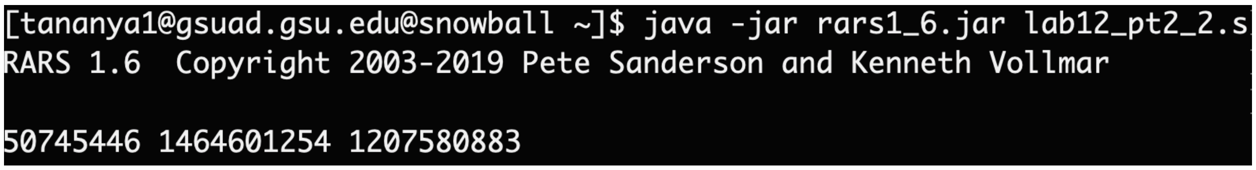
```
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

25372723 732300627 -1543695255
```



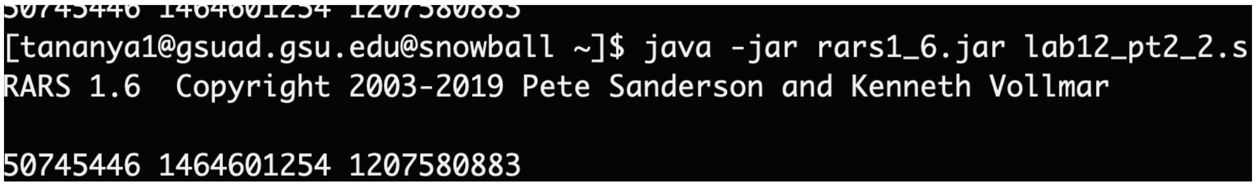
```
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

25372723 732300627 -1543695255
```



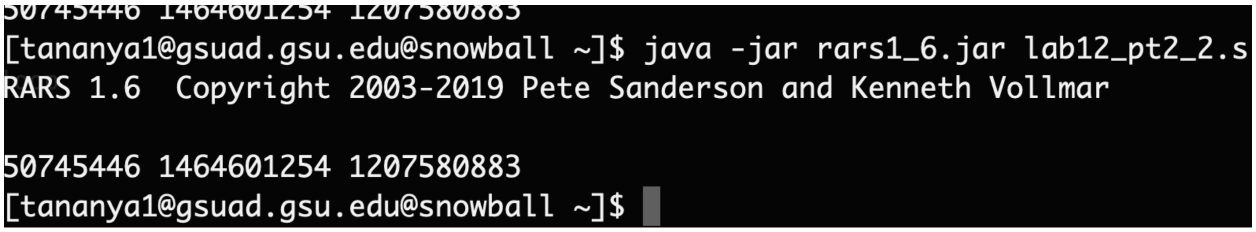
```
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

25372723 732300627 -1543695255
```



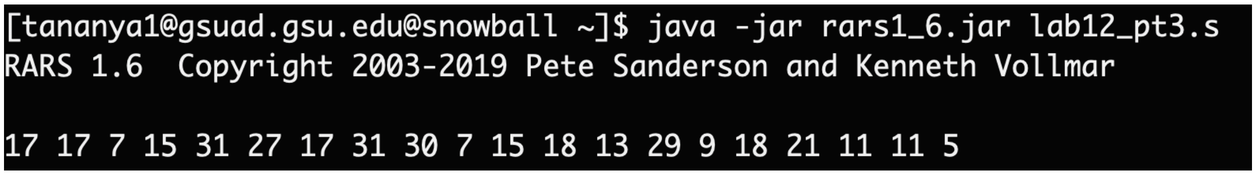
```
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2_2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

50745446 1464601254 1207580883
```



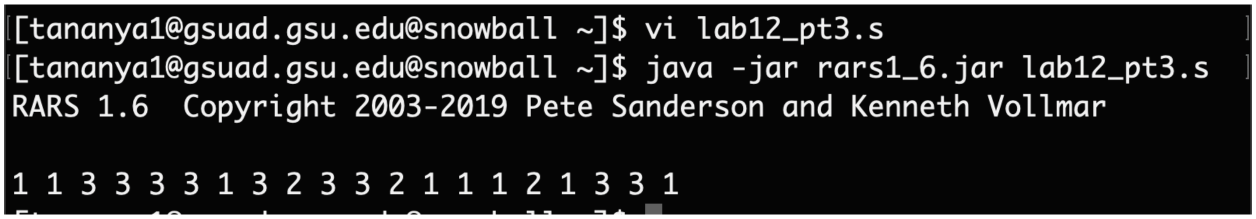
```
50745446 1464601254 1207580883
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2_2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

50745446 1464601254 1207580883
```



```
50745446 1464601254 1207580883
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2_2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

50745446 1464601254 1207580883
[tananya1@gsuad.gsu.edu@snowball ~]$
```



```
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt3.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

17 17 7 15 31 27 17 31 30 7 15 18 13 29 9 18 21 11 11 5
```



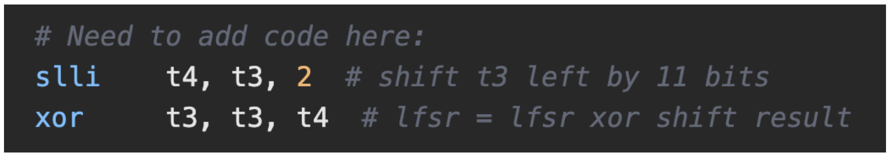
```
[tananya1@gsuad.gsu.edu@snowball ~]$ vi lab12_pt3.s
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt3.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

1 1 3 3 3 3 1 3 2 3 3 2 1 1 1 2 1 3 3 1
[tananya1@gsuad.gsu.edu@snowball ~]$
```

Questions:

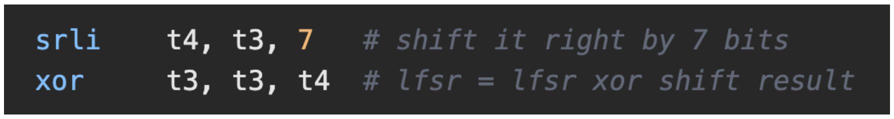
Part One:

1. **Does your code work with shifts of 2, 7, 9 as well? Note that this is from the Marsaglia paper.**

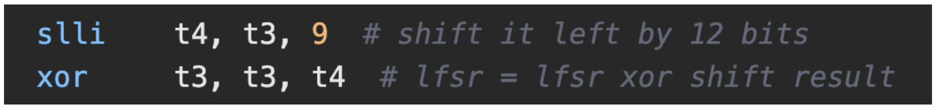
   Modifying the code to hold the shifts of 2, 7, 9 will work just as well because they all produce shift values that are compatible and capable of generating random numbers.



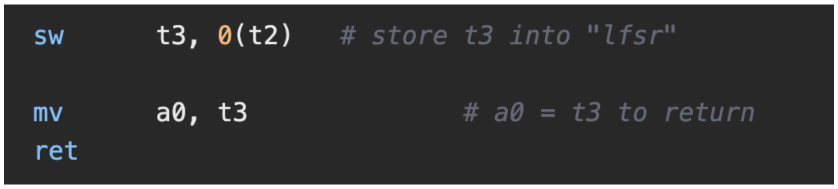
```
# Need to add code here:
slli    t4, t3, 2  # shift t3 left by 11 bits
xor     t3, t3, t4 # lfsr = lfsr xor shift result
```



```
srli    t4, t3, 7  # shift it right by 7 bits
xor     t3, t3, t4 # lfsr = lfsr xor shift result
```



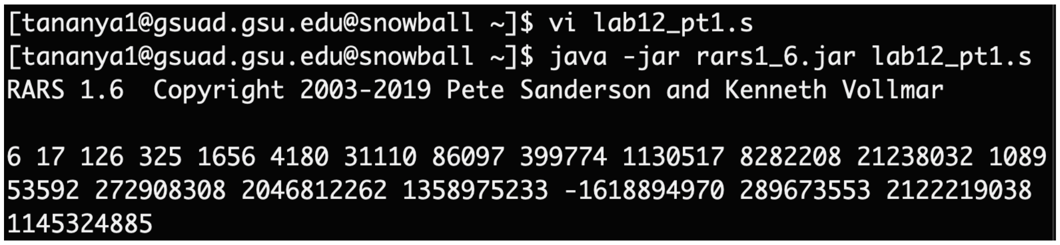
```
slli    t4, t3, 9  # shift it left by 12 bits
xor     t3, t3, t4 # lfsr = lfsr xor shift result
```



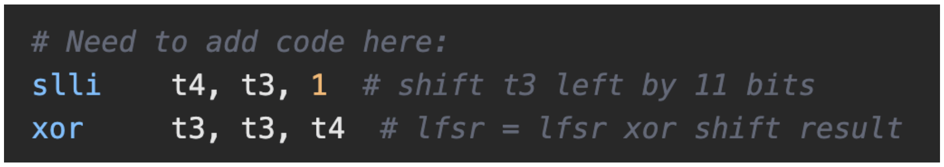
```
sw      t3, 0(t2)   # store t3 into "lfsr"

mv      a0, t3              # a0 = t3 to return
ret
```

2. **Does your code work with shifts of 1, 1, 1 as well? Note that this is not one of the triplets from the Marsaglia paper. Why do you think the triplet 1,1,1 is not as good as 2,7,9?**

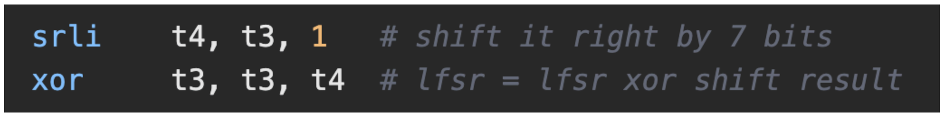
Yes, the code works with shifts of 1, 1, 1 as well but it does not give appropriate values of appropriate lengths. As shown in the screenshot below, the number lengths are off, they aren't entirely random, and have positives with negatives. The triplet 1, 1, 1 is not as good as 2, 7, 9 because the execution produces poorer quality of results than 2, 7, 9.



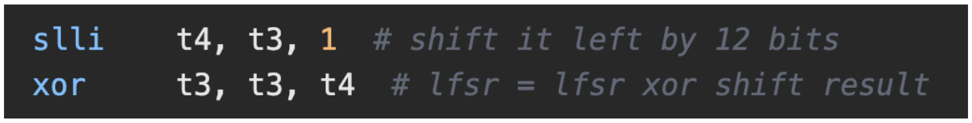
```
[tananya1@gsuad.gsu.edu@snowball ~]$ vi lab12_pt1.s
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt1.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

6 17 126 325 1656 4180 31110 86097 399774 1130517 8282208 21238032 1089
53592 272908308 2046812262 1358975233 -1618894970 289673553 2122219038
1145324885
```



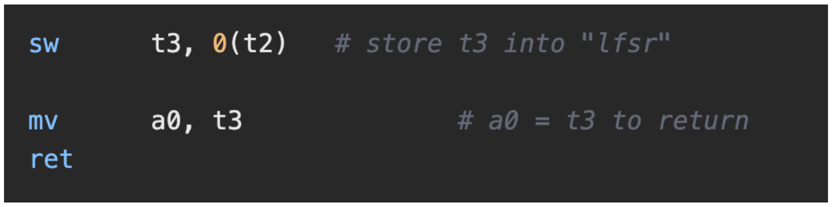
```
# Need to add code here:
slli    t4, t3, 1  # shift t3 left by 11 bits
xor     t3, t3, t4  # lfsr = lfsr xor shift result
```



```
srli    t4, t3, 1   # shift it right by 7 bits
xor     t3, t3, t4  # lfsr = lfsr xor shift result
```



```
slli    t4, t3, 1  # shift it left by 12 bits
xor     t3, t3, t4  # lfsr = lfsr xor shift result
```



```
sw      t3, 0(t2)   # store t3 into "lfsr"

mv      a0, t3                  # a0 = t3 to return
ret
```

Part Two:

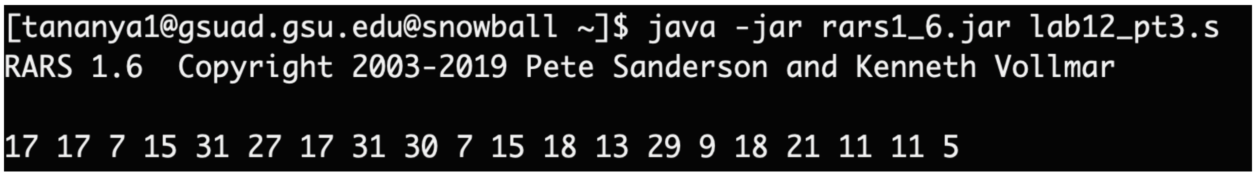
1. **Suppose that we set the seed to the initial data value in part 1 (lfsr: .word 1), but set it with "set_seed". Do you get the same values as you did in part 1? Why or why not?**

   If we set the seed to the initial data value in part 1 (lsfrL .word 1), but set it with “set_seed”, I get the same values as I did in part 1. This is because the RNG starts before the loop, giving the opportunity for lsfr to be the same, thus giving the same value results after running as from part 1.

Part Three:

1. **Does your program generate any negative values now? Why or why not?**

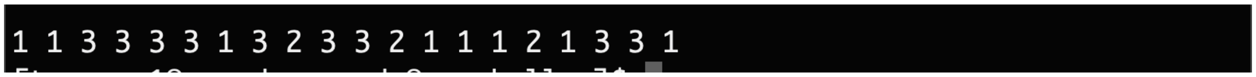
   After modifying the code with a cut off at 32 characters with AND and 0x1f, the program does not print anymore negative values. This is because the 0x1f cuts off the result at the lowest 5 bits. Thus, the resulting values as output are positive since this cut off does not accommodate negative integer values being returned.



```
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt3.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

17 17 7 15 31 27 17 31 30 7 15 18 13 29 9 18 21 11 11 5
```

2. **Suppose that we AND the value with 0x03. When you run it, you will see a lot of repeated values. Is this still a random sequence? Explain.**

   If we AND the value with 0x03 instead of 0x1f and run it, we see the following pattern:



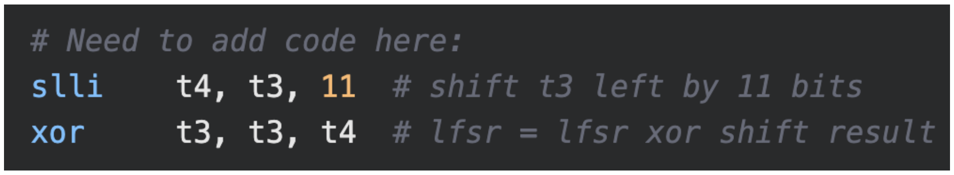
```
1 1 3 3 3 3 1 3 2 3 3 2 1 1 1 2 1 3 3 1
```

   Although there seems to be some sort of pattern, it is not a pattern. This is still a random sequence but still appears as a pattern because using 0x03 cuts off the values at the last 2 bits–which has a range of 0 to 3. Statistically, this is prone to result in returning pattern-like values since there is such a small range of numbers to print.

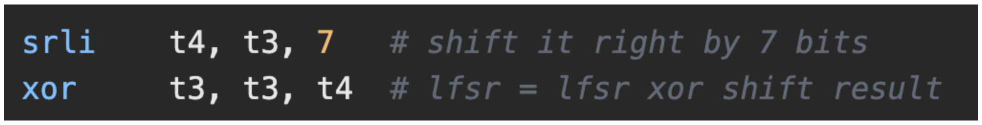
3. **Why do we limit the random values to under a power of 2? What would we need to do to limit it to the range 0 to 99?**

   The use of conditions like 0x1f and 0x03 are considerate of the power of 2. Using such values is what makes up the limit for the random values to under a power of 2. In order to limit to a range between the values 0 to 99, we need to utilize a similar condition using the power of 2 such as 0x7f because this has a similar value of 127. Then, we can cut those values off at 99 with a modulo for example.

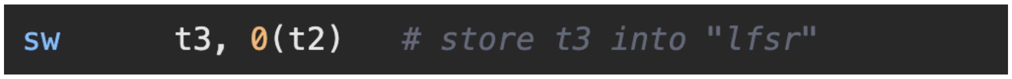
Key Code Observations



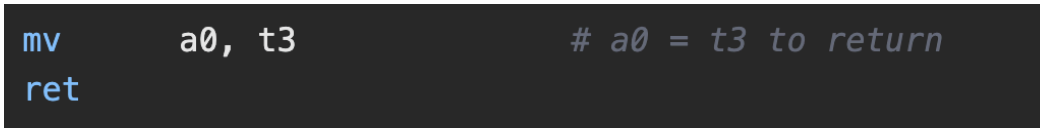
```
# Need to add code here:
slli    t4, t3, 11  # shift t3 left by 11 bits
xor     t3, t3, t4  # lfsr = lfsr xor shift result
```



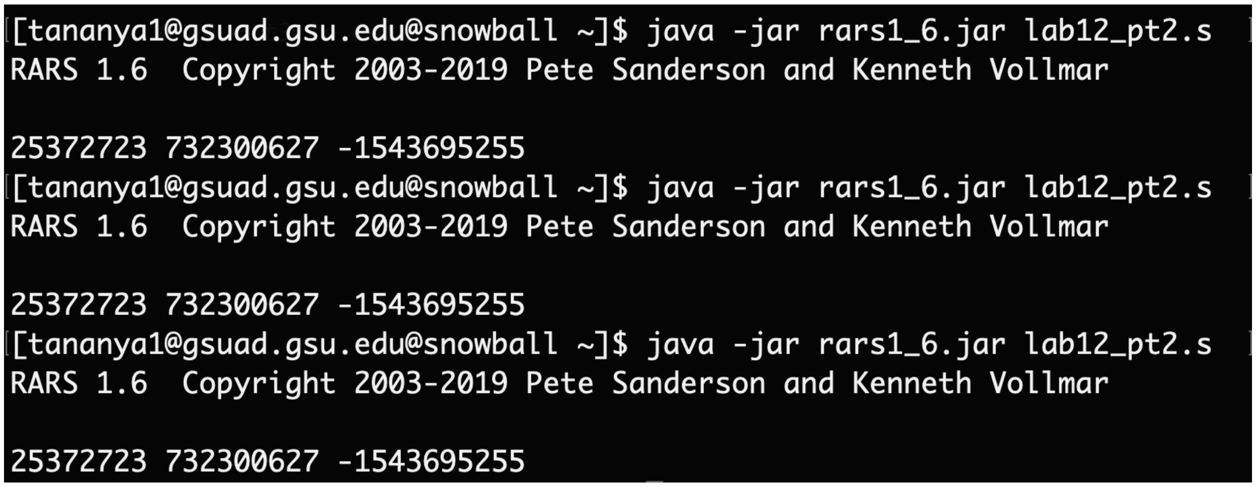
```
srli    t4, t3, 7   # shift it right by 7 bits
xor     t3, t3, t4  # lfsr = lfsr xor shift result
```



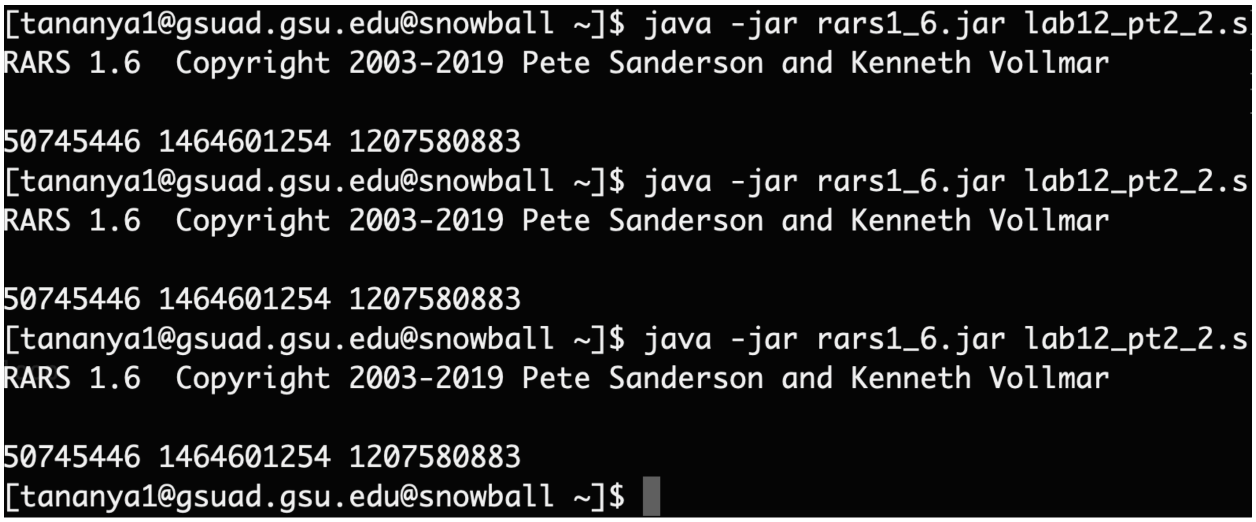
```
sw      t3, 0(t2)   # store t3 into "lfsr"
```



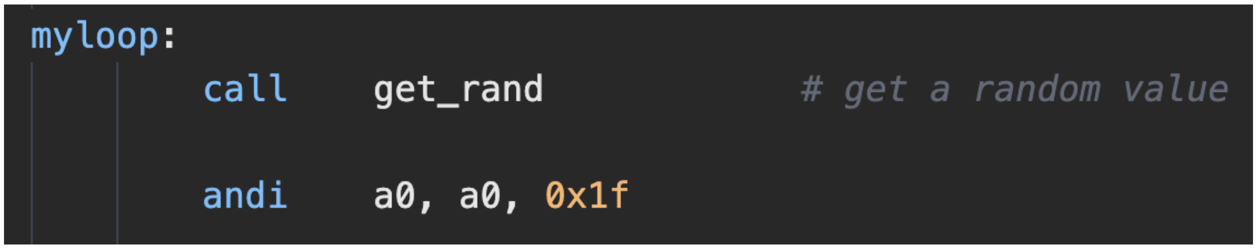
```
mv      a0, t3                  # a0 = t3 to return
ret
```



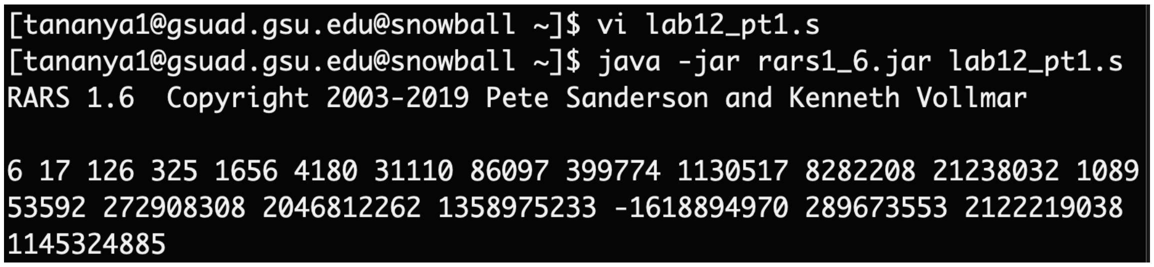
```
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

25372723 732300627 -1543695255
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

25372723 732300627 -1543695255
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

25372723 732300627 -1543695255
```



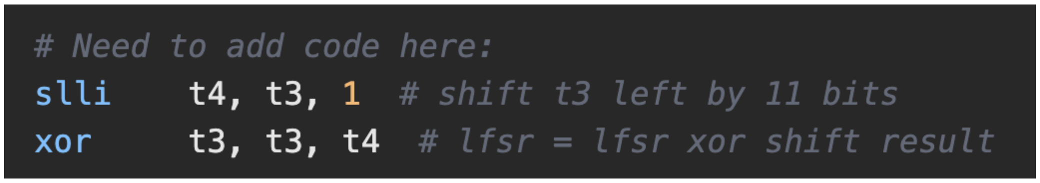
```
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2_2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

50745446 1464601254 1207580883
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2_2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

50745446 1464601254 1207580883
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt2_2.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

50745446 1464601254 1207580883
[tananya1@gsuad.gsu.edu@snowball ~]$
```



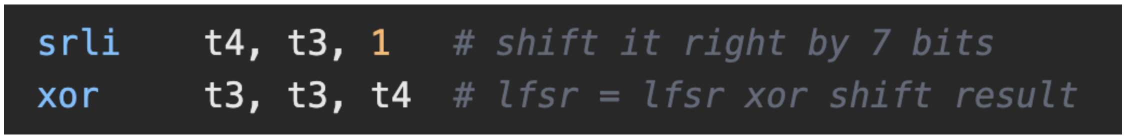
```
myloop:
        call    get_rand                # get a random value

        andi    a0, a0, 0x1f
```



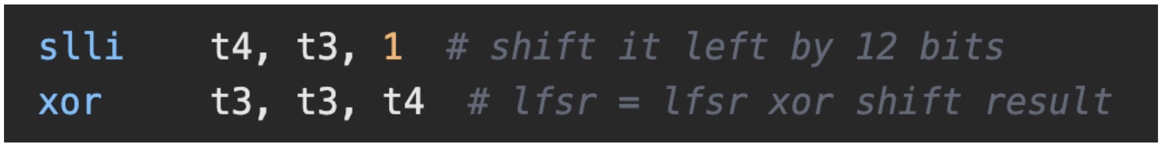
```
[tananya1@gsuad.gsu.edu@snowball ~]$ vi lab12_pt1.s
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt1.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

6 17 126 325 1656 4180 31110 86097 399774 1130517 8282208 21238032 1089
53592 272908308 2046812262 1358975233 -1618894970 289673553 2122219038
1145324885
```



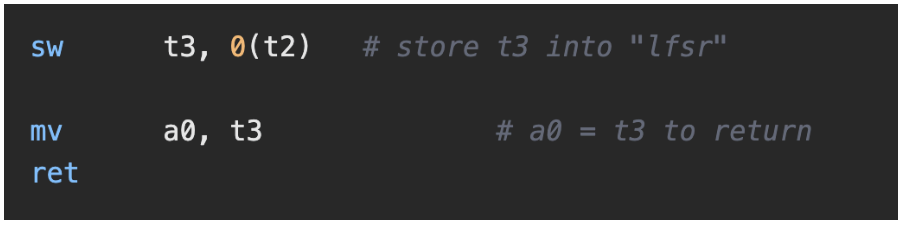
```
# Need to add code here:
slli    t4, t3, 1  # shift t3 left by 11 bits
xor     t3, t3, t4  # lfsr = lfsr xor shift result
```



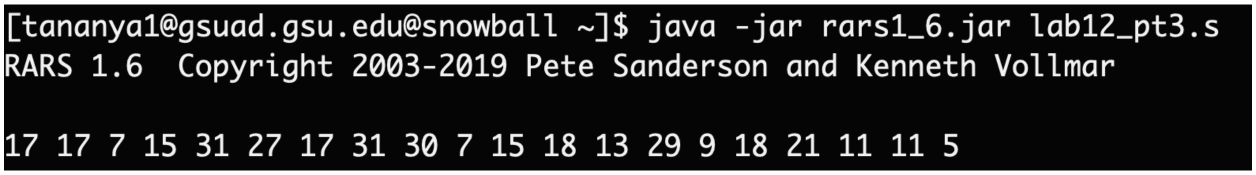
```
srli    t4, t3, 1   # shift it right by 7 bits
xor     t3, t3, t4  # lfsr = lfsr xor shift result
```



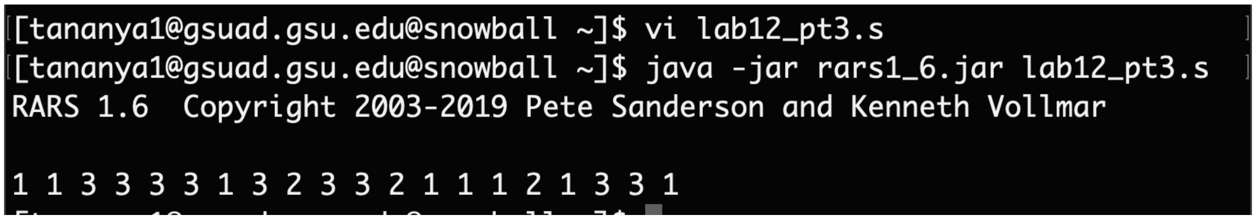
```
slli    t4, t3, 1  # shift it left by 12 bits
xor     t3, t3, t4  # lfsr = lfsr xor shift result
```



```
sw      t3, 0(t2)   # store t3 into "lfsr"

mv      a0, t3              # a0 = t3 to return
ret
```



```
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt3.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

17 17 7 15 31 27 17 31 30 7 15 18 13 29 9 18 21 11 11 5
```



```
[tananya1@gsuad.gsu.edu@snowball ~]$ vi lab12_pt3.s
[tananya1@gsuad.gsu.edu@snowball ~]$ java -jar rars1_6.jar lab12_pt3.s
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

1 1 3 3 3 3 1 3 2 3 3 2 1 1 1 2 1 3 3 1
```